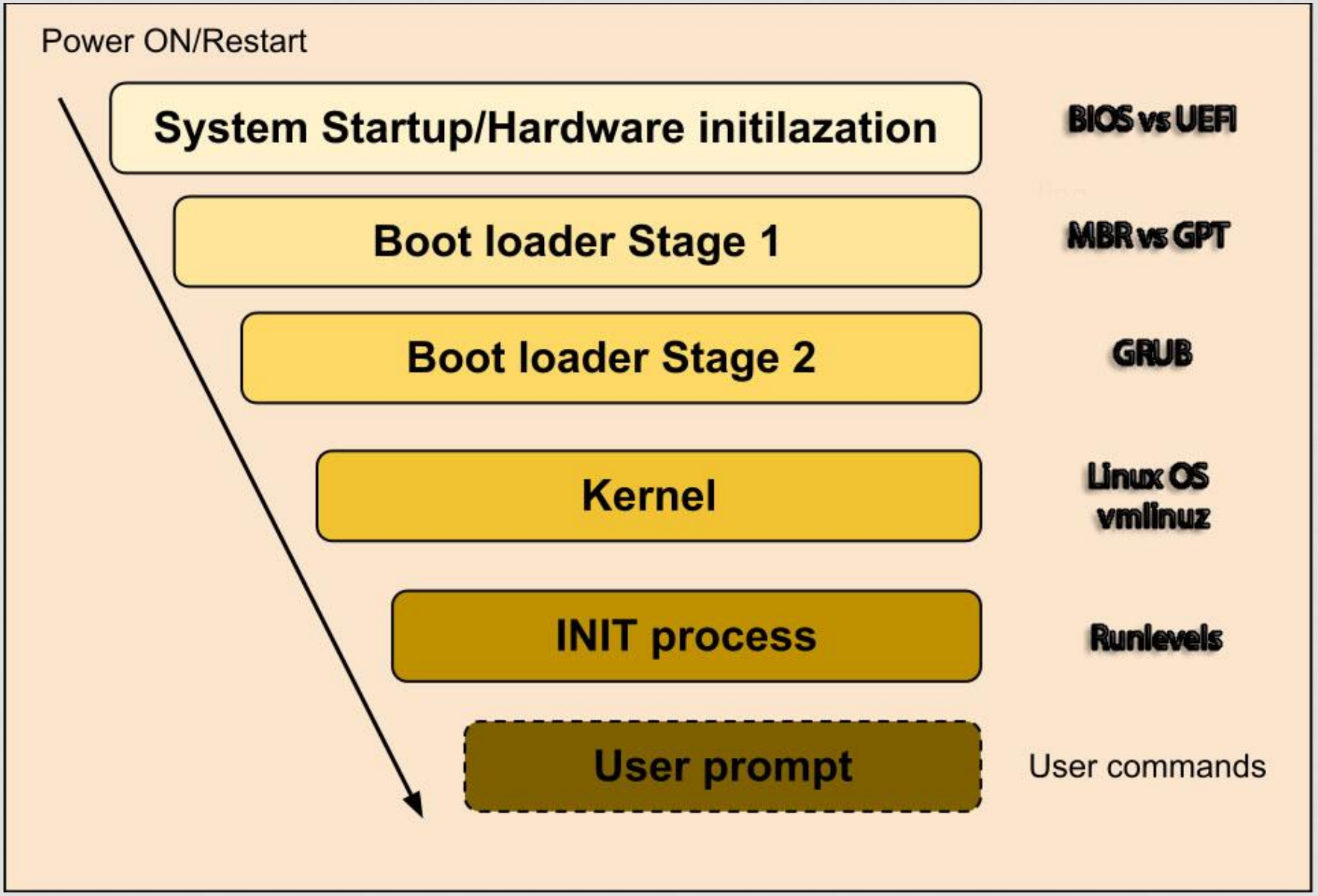
Процесс загрузки Linux


Power ON/Restart
System Startup/Hardware initilazation
BIOS vs UEFI
Boot loader Stage 1
MBR vs GPT
Boot loader Stage 2
GRUB
Kernel
Linux OS
vmlinuz
INIT process
Runlevels
User prompt
User commands

BIOS vs UEFI

BIOS

Выполняет базовое тестирование системы при включении питания POST(power-on self test)

Потом выполняется нумерация и инициализации локальных устройств.

После этих действий BIOS передает управление MBR.

UEFI

В современных системах на смену BIOS пришел UEFI (Unified Extensible Firmware Interface).

UEFI - программный интерфейс, между операционной системой и прошивкой платформы.

UEFI использует GPT (GUID Partition Table) вместо таблиц MBR.

Если размер диска превышает 2.2 TB, то без GPT уже не обойтись (MBR не может их поддерживать).

MBR

Типичная загрузка системы производиться HDD, на котором в MBR содержится первичный начальный загрузчик.

MBR представляет собой сектор размером 512 байт, который располагается в первом секторе диска (сектор 1 цилиндра 0, головка 0).

```
# dd if=/dev/sda of=MBR bs=512 count=1
# od -xa MBR
```

MBR

Первые 446 байт представляют собой первичный загрузчик, который содержит как программный код, так и текст сообщений об ошибках.

Следующие 64 байта представляют собой таблицу разделов, которая содержит запись для каждого из четырех разделов диска (по 16 байт каждая).

16 байт содержат: флаг (активный или нет), начало, конец, тип раздела (какая файловая система используется), смещение.

В конце MBR располагаются два байта, которые носят название “магического числа” (0xAA55). Это магическое число служит для целей проверки MBR.

GPT (отличия)

- В начале может содержать запись MBR.
- Поддерживает диски до 8 миллиардов террабайт
- Поддерживает 128 разделов
- Тип раздела записан 16 байтами, что позволяет точнее определять его назначение
- Имена разделов можно задавать в текстовом виде

GRUB

После первичного загрузчика (MBR/GPT) подгружается GRUB (GRand Unified Bootloader), который уже понимает что такое файловая система и может показать список имеющихся ядер и вариантов загрузки.

Эти варианты обычно определяются в следующих файлах:

- /etc/grub.conf
- /boot/grub/menu.lst
- /boot/grub/grub.cfg

GRUB

Пример конфигурации пункта в загрузочном меню:

```
menuentry 'Ubuntu, with Linux 3.8.0-26-generic'  {
        set root='(hd0,5)'
        linux   /boot/vmlinuz-3.8.0-26-generic root=/dev/sda5 ro
        initrd  /boot/initrd.img-3.8.0-26-generic
}
```

Здесь указываться образ временной файловой системы initrd.img и образ ядра vmlinuz, которому передается следующий этап загрузки.

Ядро

После того, как образ ядра оказывается в памяти, ему передается управление от загрузчика 2-й ступени (GRUB).

Если имеется образ начального RAM-диска (initrd), то программа также перемещает его в память и помечает для дальнейшего использования.

Затем вызывает само ядро, начинается его загрузка.

initrd используется самим ядром в качестве временной корневой файловой системы, пока kernel не загрузится в реальную примонтированную файловую систему.

Этот временный диск также содержит необходимые для загрузки драйверы, позволяющие получить доступ к разделам дисков и другому оборудованию.

Ядро

```
root@:~# lsinitramfs /boot/initrd.img-3.8.0-26-generic
                     | head
/boot/initrd.img-3.8.0-26-generic
.
sbin
sbin/hwclock
sbin/wait-for-root
sbin/mount.ntfs-3g
sbin/mount.ntfs
sbin/dmsetup
sbin/udevd
sbin/udevadm
```

Ядро

- Ядро монтирует файловую систему в соответствии с настройкой «root=» в фале grub.conf
- Выполняет программу /sbin/init
- Поскольку init — это первый процесс, запущенный ядром Linux, поэтому она имеет идентификатор процесса (PID) №1. Можете выполнить «ps -ef | grep init» и убедиться в этом.
- initrd — это Initial RAM Disk, он же временный диск в оперативной памяти
- initrd используется самим ядром в качестве временной корневой файловой системы, пока kernel не загрузится в реальную примонтированную файловую систему. Этот временный диск также содержит необходимые для загрузки драйверы, позволяющие получить доступ к разделам дисков и другому оборудованию
- dmesg – сообщения, которые выводило ядро

Init

После загрузки и инициализации ядра запускается первое приложение в пространстве пользователя /sbin/init, которое обращается к файлу /etc/inittab (или /etc/init/rc-sysinit.conf) для того, чтобы определить уровень выполнения (run level).

Есть следующие уровни выполнения:

- 0 — выполняются действия по выключению системы.
- 1 — однопользовательский режим (single user mode). Предназначен для восстановления системы (аналогичен Safe Mode Windows).

……

- 6 — выполняются действия по перезагрузке системы.

Init

Есть следующие уровни выполнения:

……

- 2 —В RedHat и SuSE Linux сконфигурирован как уровень выполнения 3, но без поддержки сетевых файловых систем. В Ubuntu и Debian используется как многопользовательский режим.
- 3 — многопользовательский режим (multiuser mode). Нормальный режим работы сервера.
- 4 — В Slackware Linux используется для графического входа в систему. В RedHat и SuSE Linux не сконфигурирован.
- 5 — В RedHat и SuSE Linux используется для графического входа в систему. В Slackware Linux не сконфигурирован.

……

Init

Посмотреть текущий уровень исполнения и задать дефотлный:

```
root@:~# runlevel
N 2
root@:~# vim /etc/init/rc-sysinit.conf
...
env DEFAULT_RUNLEVEL=2
...
```

Runlevel

Для каждого уровня выполнения в Ubuntu есть своя папка со скриптами:

```
root@:~# ls -ld /etc/rc*
drwxr-xr-x 2 root root 4096 Dec 31 22:48 /etc/rc0.d
drwxr-xr-x 2 root root 4096 Jan 15 13:48 /etc/rc1.d
drwxr-xr-x 2 root root 4096 Jan 15 13:48 /etc/rc2.d
drwxr-xr-x 2 root root 4096 Jan 15 13:48 /etc/rc3.d
drwxr-xr-x 2 root root 4096 Jan 15 13:48 /etc/rc4.d
drwxr-xr-x 2 root root 4096 Jan 15 13:48 /etc/rc5.d
drwxr-xr-x 2 root root 4096 Dec 31 22:48 /etc/rc6.d
-rwxr-xr-x 1 root root  420 Jul 15  2013 /etc/rc.local
drwxr-xr-x 2 root root 4096 Dec 20 16:42 /etc/rcS.d
```

Runlevel

Программа init смотрит на уровень выполнения заданный по дефолту и следует в соответствующею папку со скриптами.

```
root@:~# ls -l /etc/rc2.d/
-rw-r--r-- 1 root root 677 Jul 26  2012 README
lrwxrwxrwx 1 root root  21 Jun 18  2013 S01ramfs_mount -> ../init.d/ramfs_mount
lrwxrwxrwx 1 root root  14 Jun 17  2013 S20atop -> ../init.d/atop
lrwxrwxrwx 1 root root  18 Jun 17  2013 S20icecast2 -> ../init.d/icecast2
lrwxrwxrwx 1 root root  19 Jun 17  2013 S20memcached -> ../init.d/memcached
……..
lrwxrwxrwx 1 root root  13 Jan 15 13:48 S23ntp -> ../init.d/ntp
lrwxrwxrwx 1 root root  15 Jun 17  2013 S50rsync -> ../init.d/rsync
```

Скриптам, которые начинаются на S (startup) передается параметр start.
Скриптам, которые начинаются на K (kill) передается параметр stop.

Runlevel

```
root@:~$ cat /etc/init.d/testapp
#!/bin/bash
DAEMON_PATH="/home/antonk"
NAME=testapp
DESC="My test script"

case "$1" in
start)
        printf "%-50s" "Starting $NAME..."
        echo "START from $0 script at `date +%T` " >> $DAEMON_PATH/testappfile
;;
# ..... status  ......
stop)
        printf "%-50s" "Stopping $NAME"
        echo "STOP from $0 script at `date +%T` " >> $DAEMON_PATH/testappfile
;;
restart)
        $0 stop
        $0 start
;;
*)
        echo "Usage: $0 {status | start | stop | restart}"
        exit 1
esac
```

Runlevel

Мы создали скрипт с опциями (status | start | stop | restart).
При вызове одной из них – мы записываем вывод с указанием времени в файл /home/antonk/testappfile.
Добавим запуск скрипта на стандартных уровнях выполнения.
root@:~# update-rc.d testapp defaults
Перезагружаем систему и смотрим в файл testappfile.
root@:~# cat /home/alex/testappfile
STOP from /etc/rc6.d/K20testapp script at 05:50:17
START from /etc/rc2.d/S20testapp script at 05:50:32
Как видим, после сигнала reboot система переходит в уровень выполнения “6” и запускает скрипты начинающиеся с “K”.
Далее система стартует и переходит в стандартный уровень выполнения (уровень 2 для Ubuntu), где запускает скрипты “S”.

Runlevel

Если нужно добавить некоторый скрипт в автозагрузку – можно также использовать /etc/rc.local. Команды из этого скрипта запускается самым последним, после запуска всех скриптов из rc*.d

```
root@:~# cat /etc/rc.local
...
echo "from $0  at `date +%T` " >> /home/antonk/testappfile
exit 0
```

После загрузки – смотрим в файл.

```
root@ubuntu:~# cat testappfile
STOP from /etc/rc6.d/K20testapp script at 05:53:43
START from /etc/rc2.d/S20testapp script at 05:53:59
from /etc/rc.local  at 05:53:59
```

Как видим – команда из скрипта /etc/rc.local запустилась самой последней.

Демоны

- Скрипты из папок /etc/rc#.d/ являются (почти всегда) символическими ссылками на скрипты из /etc/init.d/
- В этой папке лежат скрипты для контроля большинства сервисов.

- Пример использования

/etc/init.d/apache start

/etc/init.d/sshd restart

/etc/init.d/apache stop

Альтернатива

- Стандартный механизм запуска через набор скриптов соответствующего уровня выполнения – сугубо последовательный.
- Альтернативой является upstart – система загрузки, основанная на обработке событий. Также данная система запускает скрипты по возможности параллельно. Используется в Ubuntu.

Upstart

Система upstart при запуске, процессом init генерируется событие **startup** - старт системы, а событие **shutdown** - при выключении системы.

Другое ключевое событие - это **ctrlaltdel**, которое указывает, на то, что вы нажали клавиши Ctrl-Alt-Delete.

В соответствии с событием генерируемым процессом init, обрабатываются конфигурационные фалы (*.conf) из каталога /etc/init/ .

В этом, собственно, и заключается вся работа **upstart**. Для обратной совместимости с существует файл /etc/init/rc.conf, который запускает скрипты из стандартных папок.

Пример

```
# myservice - myservice job file

description "my service description"
author "Me <myself@i.com>"

# When to start the service
start on runlevel [2345]

# When to stop the service
stop on runlevel [016]

# Automatically restart process if crashed
respawn

# Essentially lets upstart know the process will detach itself to the background
expect fork

# Run before process
pre-start script
    [ -d /var/run/myservice ] || mkdir -p /var/run/myservice
    echo "Put bash code here"
end script

# Start the process
exec myprocess
```

Сервисы

- Для контроля служб используется команда service (работает и со стандартными службами и службами uptime)

service apache start

service sshd stop

service apache restart

- Для контроля служб uptime используйте команду initctl

Окончание

- Демон консоли (getty) запускает программу login, которая выводит предложение ввести имя пользователя. Получив это имя, **login** обращается к файлу **/etc/passwd** за получением необходимых данных об этом пользователе - о его идентификаторе, идентификаторе группы, домашнем каталоге и о том, какую оболочку для него запускать.
- После выводится запрос на ввод пароля пользователя. Программа **login** считывает его, криптует и сравнивает результат с тем, что лежит в соответствующей строке файла **etc/shadow**.
- Если пользователь ввел правильный пароль, программа login, запускает оболочку.